B

Intrusion Detection Systems (IDS): A Survey of Existing Systems and A Proposed Distributed IDS Architecture

Steven R. Snapp[1], *James Brentano*[1], *Gihan V. Dias*[1], *Terrance L. Goan*[1], *Tim Grance*[2],
L. Todd Heberlein[1], *Che-Lin Ho*[1], *Karl N. Levitt*[1], *Biswanath Mukherjee*[1], *Douglass L. Mansur*[3],
Kenneth L. Pon[3] *and Stephen E. Smaha*[4]

1. Division of Computer Science, University of California, Davis, California 95616
2. United States Air Force Cryptologic Support Center, San Antonio, Texas 78243
3. Lawrence Livermore National Laboratory, Livermore, California 94550
4. Haystack Laboratories, Inc., Austin, Texas 78729

CSE - 91 - 7

Abstract

The study of providing security in computer networks is a rapidly growing area of interest because the network is the medium over which most attacks or intrusions on computer systems are launched. One approach to solving this problem is the *intrusion-detection* concept, whose basic premise is that not only abandoning the existing and huge infrastructure of possibly-insecure computer and network systems is impossible, but also replacing them by totally-secure systems may not be feasible or cost effective. Previous work on intrusion-detection systems were performed on stand-alone hosts and on a broadcast local area network (LAN) environment. The focus of our present research is to extend our network intrusion-detection concept from the LAN environment to arbitrarily wider areas with the network topology being arbitrary as well. The generalized distributed environment is heterogeneous, i.e., the network nodes can be hosts or servers from different vendors, or some of them could be LAN managers, like our previous work, a network security monitor (NSM), as well. The proposed architecture for this distributed intrusion-detection system consists of the following components: a host manager (viz. a monitoring process or collection of processes running in background) in each host; a LAN manager for monitoring each LAN in the system; and a central manager which is placed at a single secure location and which receives reports from various host and LAN managers to process these reports, correlate them, and detect intrusions.

1. Introduction

Intrusion detection and *network security* are becoming increasingly more important in today's computer-dominated society. As more and more sensitive information is being stored on computer systems and transferred over computer networks, there are an increasing number of attacks on these systems in an attempt to steal, destroy or corrupt that information. While most computer systems attempt to prevent unauthorized use by some kind of access control mechanism, such as passwords, there are several factors that make it very difficult to keep these attackers from eventually gaining entry into a system [2,6]. Most computer systems have some kind of security flaw that may allow outsiders (or legitimate users) to gain unauthorized access to sensitive information. In most cases, it is not practical to replace such a flawed system with a new, more secure system. It is also the case that it is very difficult, if not impossible, to develop a completely-secure system. Even a supposedly-secure system can still be vulnerable to insiders misusing their privileges, or it can be compromised by improper operating practices. While many existing systems may be designed to prevent specific types of attacks, other methods to gain unauthorized access may still be possible. Due to the tremendous investment already made into the existing infrastructure of *open* (and possibly insecure) communication networks, it is infeasible to deploy new, secure, and possibly *closed* networks [5]. Since the event of an attack should be considered inevitable, there is an obvious need for mechanisms that can detect outsiders attempting to gain entry into a system, that can detect insiders misusing their system privileges, and that can monitor the networks connecting all of these systems together.

Intrusion Detection Systems (IDS) are based on the principle that an attack on a computer system (or network) will be noticeably different from normal system (or network) activity. An intruder to a system (possibly masquerading as a legitimate user) is very likely to exhibit a pattern of behavior different from the normal behavior of a legitimate user. The job of the IDS is to detect these abnormal patterns by analyzing numerous sources of information that are provided by the existing systems. The two major methods used to detect intrusions are *statistical analysis* and rule-based *expert system analysis*. The statistical method attempts to define *normal* (expected) behavior, while the expert system defines *proper* behavior. The expert system also searches for breaches of policy. If the IDS notices a possible attack using either of these methods, then a *System Security Officer* (SSO) is notified. The SSO may then take some action against the aggressor.

The audit trail records provided by a computer system are the main source of information regarding behavior for that system. The IDS must use these records in its analysis of the expected and proper behavior for that system. The two major problems with audit trail analysis are exactly what kind of activity should be audited (i.e., auditable events), and what to do with the huge amount of information that is produced. Since auditing was initially designed for accounting purposes rather than for system security, many desired auditable events may be unavailable to the security system. The audit trail is designed and provided by the vendor of the computer system in question. The amount of information generated by the audit trail mechanism for even a single user can be tremendous, possibly in excess of 10 megabytes of data per user per day [16]. Processing such a huge amount of information, viz. the collection, storage and timely analysis of the records, can become problematic, since overall system performance may become adversely affected. Another continuing source of disagreement is the idea of a so-called *canonical audit record*. The question is whether or not to standardize the kind of information that should be contained in an audit record.

One means of detecting anomalous behavior is to monitor statistical measures of user activities on the system. A popular way to monitor statistical measures is to keep *profiles* of legitimate user activities [2,6]. These profiles may include such items as login times, CPU usage, favorite editor and compiler, disk usage, number of printed pages per session, session length, error rate, etc. (T.F. Lunt et al., [7] presents a comprehensive list of possible measures.) The IDS will then use these profiles to compare current user activity with past user activity. Whenever a current user's activity pattern falls outside certain pre-defined thresholds, the behavior is considered anomalous. Legitimate behavior that is flagged as intrusive is defined to be a *false alarm*. A major problem with the statistical method is determining exactly what activities and statistical measures provide the highest detection rate and lowest false alarm rate for a particular system. Those statistics that detect an attack on a computer system may differ from system to system depending on the system and its environment; so the measures must be tailored for each particular system. It may also be the case that a particular activity may not be threatening by itself, but when aggregated with other activities, it may constitute an attack. These statistical profiles must be adaptive, i.e., they must be updated regularly, since users may be constantly changing their behavior.

Another means of detecting possible attacks on a computer system is by using a rule-based expert system analysis method. The expert system will analyze the audit trail records, and it will try to determine attacks based on the information contained in the rule base. The expert system is able to pose sophisticated queries to the rule base to answer conditional questions based on sets of events [16]. The main problem with this method is determining exactly what the rules should be, and what kinds of attacks can be detected using this method. As an example, the rule base may contain rules describing known attack methods and signatures, known system flaws, expected system behavior, and the site-specific security policy.

Network intrusions such as eavesdropping, illegal remote accesses, remote break-ins, inserting bogus information into data streams and flooding the network to reduce the effective channel capacity are becoming more common. While monitoring network activity can be difficult because of the great proliferation of (un)trusted open networks, the task is not impossible. The same type of system that can monitor a single host machine may also be able to monitor a *local area network* (LAN) just as effectively [5]. The key here, again, is to attempt to define normal network activity based on past performance or experience, as well as proper network behavior.

Section 2 of this paper presents a review of several existing Intrusion Detection Systems. Section 3 presents a discussion of our current work, the Network Security Monitor (NSM), and Section 4 presents a discussion of a proposed Distributed Intrusion Detection project that is currently under investigation.

2. Review of Present Intrusion Detection Systems

There are several intrusion detection systems that are currently in operation. This section presents a review of these known systems.

2.1. Computer Security Monitor (CSM)

2.1.1. Overview

The Technology Information Systems Program at Lawrence Livermore National Laboratory has designed and implemented a prototype Computer Security Monitor (CSM) [9]. The device has been designed to be hardware and operating system independent and be broad enough to be useful for any host system that can generate an audit trail. Currently, the prototype monitors audit records produced from both an IBM MVS/SP operating system and SUN's SunOS4.0 version of the UNIX operating system.

Besides incorporating many of the characteristics of the ideal classical intrusion-detection system (such as real-time monitoring, operating system independence, and addressing the issues of insider threat and data aggregation), the CSM contains several unique design features that include simultaneous and independent statistical and deterministic views, configurable levels of machine interdiction, severity indicators, facilities that provide redundancy for multi-level secure systems, and a rich, mature graphical user interface.

2.1.2. System Organization

The CSM is a computer system that is attached to one or more target hosts via two dedicated communications lines: an input line transports audit records from the target host to the CSM; an output line carries interdiction commands from either the SSO or (if so configured) by the CSM. End-to-end encryption mechanisms insure the integrity of communications. Because the CSM is physically isolated from the users of the target computer system, it is less likely to suffer the unwanted effects of tampering. In addition, the information auditing activities of the monitor would not impinge heavily on the resources of the target machine. CSM components on the target mainframe are largely target system dependent. The remaining elements are largely target system independent. The latter subsystem was designed to provide a security monitor toolkit, whereby the SSO can select and configure interfaces, optional system runtime parameters, and site-specific, intrusion-detection characteristics to be monitored and profiled.

The CSM introduces several new concepts and facilities to add-on security. A user on the target machine can be simultaneously examined against his/her statistical profiles on one CSM window, while on another window deterministic rules set up by the SSO independently monitor adherence to discretionary access and critical resource controls.

2.1.3. System Operation

The Statistical Analysis Window examines user features such as terminal location, time of use, and transaction frequency to determine whether they fall within permissible ranges of a user's statistical profile. Audit streams that are out of range cause the CSM to signal that an *abnormality* has occurred. Abnormalities indicate that unusual, although not necessarily prohibitive, behavior has occurred.

The CSM Reportable Events Window displays noncompliance to site-specific deterministic policies and rules. The CSM allows the SSO to set up critical resource sets against which accesses (failed and/or successful) can be regarded as either *notables* or *violations*. A notable occurs when the subject has the authority to perform the associated action; however, the resource set accessed is of such high criticality or sensitivity that the access must be noted. A violation, on the other hand, reports the access (or attempted access) of a resource that has been defined to the CSM as prohibited. For example, the SSO might specify that only those users in the system administration group may write the password file without using the password command that changes a user's own password. Furthermore, when an administrator does so, the SSO may also specify that the CSM should report it as a notable. A user outside the system administration group would be guilty of a violation if that user used an editor to write the password file. A critical resource set can be any set of host system objects; accesses can be read and/or execute as well as write.

The System Status/Activity Window shows the status of both the monitored host system(s) and the CSM monitoring system. This window is especially useful during periods in which the CSM notices very little anomalous activity because the SSO can, with a single glance, consult it to check if audit records are still being received and processed.

Numerous popup windows are available to assist the SSO in the investigation of an anomalous user session. For example, a mouse click into one of the cells under the heading "Abnormal Transaction" will list a user's abnormal transactions and enable the SSO to examine the user's transaction profile. Popups essentially generate canned queries into the CSM database. The SSO can also create specific queries, as required.

2.2. ComputerWatch

2.2.1. Overview

The ComputerWatch audit trail analysis tool provides a significant amount of audit trail data reduction and some limited intrusion-detection capability [3]. The amount of data viewed by a system security officer is reduced while minimizing the loss of any informational content. The tool reduces the amount of data by providing a mechanism for examining different views of the audit data based on information relationships.

ComputerWatch, designed for the System V/MLS operating system, was written to assist, not replace, the SSO. The tool uses an expert system approach to summarize security sensitive events and to apply rules to detect anomalous behavior. It also provides a method for detailed analysis of user actions in order to track suspicious behavior.

2.2.2. System Organization

Audit trail records can be analyzed either by the SSO interactively, or in batch mode for later review, i.e., ComputerWatch does no real-time analysis of events.

There are three levels of detection statistics, viz. system, group, and user. Statistical information for system-wide events are provided in a summary report. Statistical information for user-based events are provided by detection queries. Statistical information for group-based events will be a future enhancement.

2.2.3. System Operation

ComputerWatch provides a System Activity Summary Report for the SSO. This report contains summary information describing the security-relevant activities occurring on the system. The report can indicate what types of events need closer examination. The SSO can also perform his/her own analysis on the data.

Expert system rules are used to detect anomalies or simple security breaches. The rules are fired when an equation is satisfied and when the rules in its predecessor list have been fired as well.

The detection queries that are provided have been designed to assist the SSO in detecting "simple" system security breaches. These security breaches may involve intrusion, disclosure, or integrity subversion. The detection queries display similar security-relevant system activities as those that are described in the summary report, but at a user level. A SQL-based query language is provided to allow the SSO the capability to design custom queries for intrusion-detection.

2.3. Discovery

2.3.1. Overview

Discovery is an expert system developed by TRW for detecting unauthorized accesses of its credit database [12, 13]. The Discovery system itself is written in COBOL, while the expert system is written in an AI shell. Both run on IBM 3090's. Their goal is not to detect attacks on the operating system, but to detect abuses of the application (i.e., the credit database).

2.3.2. System Organization

TRW runs a database containing the credit histories of 133 million consumers. It is accessed over 400,000 times a day using 150,000 different access codes, many of which are used by more than one person. The database is accessed in three ways: on-line access by TRW customers who query consumers' credit information, monthly updates from accounts receivable data received on magtape, and modifications to correct errors and inaccuracies. The Discovery system currently examines each of these processes for unauthorized activity.

Discovery is a statistical inference system which looks for patterns in the input data. Its targets include hackers, private investigators and criminals. It is designed to detect three types of undesired activity, viz. access by unauthorized users, unauthorized activities by authorized users, and invalid transactions. Processing of the audit data is performed in daily batches.

2.3.3. System Operation

1) Customer Inquiries. Discovery's processing sequence is as follows. First, records with invalid formats are discarded. Valid records are then sorted and processed by a pattern recognition module. Inquiries are compared to both the standard inquiry profile and a model of illegitimate access. Access codes which are suspected of having been misused can also be flagged for tighter scrutiny.

The system produces a *user profile* for each customer by type-of-service and access method. These profiles are updated daily. The system generates statistical patterns based on the variables in each inquiry (e.g., presence or absence of a middle initial), access characteristics (e.g., time of day), and characteristics of a credit record (e.g., geographic area).

Each variable has a tolerance, established by accumulated patterns, within which the daily activity should fall. Three types of comparison are made: each inquiry with the global pattern, each subscriber's daily pattern with the global pattern, and a subscriber's pattern with an industry pattern. The system's output is an exception data file that lists the reasons for the exceptions, as well as a report module. Investigative data is also stored in a database and may be retrieved using a 4th generation query language.

Initial production runs of the expert system produced large numbers of exceptions. Some of these were traced to variations due to time of day, etc. Heuristics based on analysis of actual cases are also being included in the expert system.

2) **Database Update.** Several factors in the incoming data from customers are measured by a COBOL program. Data are entered into the database only if statistical comparisons with previously reported data are within a pre-defined tolerance. Data which are rejected by the statistical analysis are submitted to an expert system for further validation, and are entered into the database if passed by the expert system.

3) **Database Maintenance.** The credit database may be modified by TRW operators to correct errors and inaccuracies. An expert system designed to monitor the maintenance process was under development (in 1988). This system will perform statistical analysis of maintenance transactions and analyze each credit record's maintenance history.

Discovery has detected and isolated unauthorized accesses to the database, masqueraders, and invalid inquiries. It has also provided investigators with concise leads on illegitimate activity. Several of the deviations that have been discovered were found to be caused by customers changing their access methods and systems. The expert system allows TRW to apply a consistent security policy in the update process. A beneficial side-effect of the system is the compilation of purchasing patterns for each customer, which is useful for marketing purposes.

2.4. HAYSTACK

2.4.1. Overview

HAYSTACK is a system for helping Air Force Security Officers detect misuse of Unisys 1100/2200 mainframes used at Air Force Bases for routine "unclassified but sensitive" data processing [11]. HAYSTACK software reduces voluminous system audit trails to short summaries of user behaviors, anomalous events, and security incidents. This reduction enables detection and investigation of intrusions, particularly by insiders (authorized users).

In addition to providing audit trail data reduction, HAYSTACK attempts to detect several types of intrusions: attempted break-ins, masquerade attacks, penetration of the security system, leakage of information, denial of service, and malicious use.

HAYSTACK's operation is based on behavioral constraints imposed by official security policies and on models of typical behavior for user groups and individual users.

2.4.2. System Organization

The HAYSTACK system consists of two program clusters, one executing on the Unisys 1100/2200 mainframe, and the other executing on a 386-based PC running MS-DOS and the ORACLE database management system. Data is transferred from the mainframe to the PC by magnetic tape or electronic file transfer over a communications line.

The preprocessor portion of HAYSTACK that runs on the mainframe is a straightforward COBOL application that selects appropriate audit trail records from the Unisys proprietary audit trail file as input, extracts the required information, and reformats it into a standardized format for processing on the PC.

Software on the PC was written in C, Embedded SQL, and ORACLE tools. It processes and analyzes the audit trail files, helps the Security Officer maintain the databases that underlie HAYSTACK, and gives the Security Officer additional support for his/her investigations.

2.4.3. System Operation

HAYSTACK helps its operator detect intrusions (or misuse) in three different ways.

1) **Notable Events.** HAYSTACK highlights notable single events for review. Events that modify the security system are reported, along with explanatory messages. This includes both successful and unsuccessful events that affect access controls, userids, and groupids.

2) **Special Monitoring.** HAYSTACK's operator may tag particular security subjects and objects as requiring special monitoring. This is analogous to setting an alarm to go off when a particular userid is active, or when a particular file or program is accessed. This alarm may also increase the amount of reporting of the user's activity.

3) **Statistical Analysis.** HAYSTACK performs two different kinds of statistical analysis. The first kind of statistical analysis yields a set of suspicion quotients. These are measures of the degree to which the user's aggregate session behavior resembles one of the target intrusions that HAYSTACK is trying to detect.

About two dozen features (behavioral measures) of the user's session are monitored on the Unisys system, including time of work, number of files created, number of pages printed, etc. Given a list of the session features whose values were outside the expected ranges for the user's security group, plus the estimated significance of each feature violation for detecting a target intrusion, HAYSTACK computes a weighted multinomial suspicion quotient that the session resembles a target intrusion for the user's security group. The suspicion quotient is therefore a measure of the anomalousness of the session with respect to a particular weighting of features. HAYSTACK emphasizes that such suspicions are not "smoking guns", but are rather hints or hunches to the Security Officer that may warrant further investigation.

The second kind of statistical analysis detects variation within a user's behavior by looking for significant changes (trends) in recent sessions compared to previous sessions.

2.5. Intrusion-Detection Expert System (IDES)

2.5.1. Overview

The Intrusion-Detection Expert System (IDES) developed at SRI International is a comprehensive system that uses complex statistical methods to detect atypical behavior, as well as an expert system that encodes known intrusion scenarios, known system vulnerabilities, and the site-specific security policy [6, 7].

The overall goal of IDES is to provide a system-independent mechanism for the *real-time* detection of security violations. These violations can be initiated by outsiders who attempt to break into a system or by insiders who attempt to misuse their privileges. IDES runs independently on its own system (currently a Sun workstation) and processes the audit data received from the system being monitored.

2.5.2. System Organization

The IDES prototype uses a subject's historical profile of activity to determine whether it's current behavior is normal with respect to past or acceptable behavior. Subjects are defined as *users, remote hosts*, or *target systems*. A profile is a description of a subject's normal (i.e., expected) behavior with respect to a set of intrusion-detection measures. IDES monitors target system activity as it is recorded in audit records generated by the target system. Due to the fact that these profiles are updated daily, IDES is able to adaptively learn a subjects' behavior patterns; as users alter their behavior, the profiles change to reflect the most recent activity. Rather than storing the tremendous amount of audit data, the subject profiles keep only certain statistics such as frequency tables, means, and covariances.

IDES also includes an expert-system component that is able to describe suspicious behavior that is independent of whether a user is deviating from past behavior patterns. The expert system contains rules that describe suspicious behavior based on knowledge of past intrusions, known system vulnerabilities, or the site-specific security policy.

The IDES comprehensive system is considered to be *loosely coupled* in the sense that the decisions made by the two components are independent. While the two components share the same source of audit records and produce similar reports, their internal processing is done separately. The desired effect of combining these two separate components is a complementary system in which each approach will help to cover the vulnerabilities of the other.

2.5.3. System Operation

The system has two major components as discussed below.

1) The Statistical Anomaly Detector (IDES/STAT). In order to determine whether or not current activity is atypical, IDES/STAT uses a deductive process based on statistics. The process is controlled by dynamically-adjustable parameters that are specific to each subject. Audited activity is described by a vector of intrusion-detection variables that correspond to the measures recorded in the profiles. As each audit record arrives, the relevant profiles are retrieved from the knowledge base and compared with the vector of intrusion-detection variables. If the point defined by the vector of intrusion-detection variables is sufficiently far from the point defined by the expected values stored in the profiles, with respect to the historical covariances for the variables stored in the profiles, then the record is considered anomalous.

The procedures are not only concerned with whether an audit variable is out of range, but also with whether an audit variable is out of range relative to the values of the other audit variables. IDES/STAT evaluates the total usage pattern, not just how the subject behaves with respect to each measure considered singly. The actual statistical algorithms are considered beyond the scope of this paper.

2) The Expert System. The IDES expert system will make attack decisions based on information contained in the rule base regarding known attack scenarios, known system vulnerabilities, site-specific security information, and expected system behavior. It will, however, be vulnerable to intrusion scenarios that are not described in the knowledge base.

The expert system component is a rule-based, forward-chaining system. A Production-Based Expert

System Tool (PBEST) has been used to produce a working system. The PBEST translator is used to translate the rule-base into C language code, which actually improves the performance of the system over using an interpreter. The current rule-base consists of 25 rules, and has a steady state processing capability of approximately 50 audit records per second. As the size of the rule-base increases, the processing time will also increase since the functions that implement the rules must search longer lists.

2.6. Information Security Officer's Assistant (ISOA)

2.6.1. Overview

The Information Security Officer's Assistant (ISOA) is a real-time network and host security monitor implemented on a UNIX-based workstation that supports automated as well as interactive audit trail analysis [16, 17]. This monitor provides a system for the timely correlation and merging of disjoint details into an assessment of the current security status of users and hosts on a network. The audit records, which are indications of actual events, are correlated with known indicators (i.e., expected events) organized in hierarchies of concern, or security status.

The ISOA's analysis capabilities include both statistical as well as expert system components. These cooperate in the automated examination of the various concern levels of data analysis. As recognized indicators (sets of indicators) are matched, concern levels increase and the system begins to analyze increasingly detailed classes of audit events for the user or host in question.

2.6.2. System Organization

Monitoring events that do not constitute direct violation of the security policy require a means to specify expected behavior on a user and host basis. The expected behavior can be represented in profiles that specify thresholds and associated reliability factors for discrete events. The observed events can then be compared to expected measures, and deviations can be identified by statistical checks of expected versus actual behavior. ISOA profiles also include a historical abstract of monitored behavior (e.g., a record of how often each threshold was violated in the past), and inferences that the expert system has made about the user. Note also that hosts as well as individual users are monitored.

Events that cannot be monitored by examining thresholds make it necessary to effect a higher order analysis that is geared towards correlating and resolving the meaning of diverse events. The expert system analysis component can specify the possible relations and implied meaning of diverse events using the rule-base. Where statistical measures can quantify behavior, the rule-based analysis can answer conditional questions based on sets of events.

2.6.3. System Operation

The underlying processing model of the ISOA consists of a hierarchy of concern levels constructed from indicators. Analysis is structured around these indicators to build a global view of the security status for each monitored user and host. The indicators allow a modeling and identification of various classes of suspicious behavior, such as aggregator, imposter, misfeasor, etc.

Two major classes of measures are defined, *real-time* and *session*. The real-time measures require immediate analysis, while session measures require (at minimum) start-of-session and end-of-session analysis.

ISOA supports two classes of anomaly detection, *preliminary* and *secondary*. Preliminary anomaly detection takes place during the collection of the audit data (i.e., in real-time). Pre-determined events trigger an investigation of the current indicator or event of interest. If further analysis is warranted, the current parameters are checked against the profiles for real-time violations or deviations from expected behavior.

Secondary anomaly detection is invoked at the end of a user login session or when required for resolution. The current session statistics are checked against the profiles, and session exceptions are determined.

When the expert system is notified that the state of indicators has changed significantly, it attempts to resolve the meaning of the current state of indicators. This is done by evaluating the appropriate subset of the overall rule-base, which consists of a number of individual rules that relate various indicator states with each other and with established threat profiles. The end result of anomaly resolution is presented to the ISO in the form of graphical alert, advice, and explanation as to why the current security level is appropriate.

2.7. Multics Intrusion Detection and Alerting System (MIDAS)

2.7.1. Overview

The Multics Intrusion Detection and Alerting System (MIDAS) is an expert system which provides real-time intrusion and misuse detection for the National Computer Security Center's networked mainframe, Dockmaster, a Honeywell DPS-8/70 Multics computer system [10].

MIDAS has been developed to employ the basic concept that statistical analysis of computer system activities can be used to characterize normal system and user behavior. User or system activity that deviates beyond certain bounds should then be detectable.

2.7.2. System Organization

MIDAS consists of several distinct parts. Those implemented on Dockmaster itself include the command monitor, a preprocessor, and a network-interface daemon. Those that are installed on a separate Symbolics Lisp machine include a statistical database, a MIDAS knowledge base, and the user interface.

The command monitor captures command execution data that is not audited by the Multics systems, while the preprocessor transforms Dockmaster audit log entries into a canonical format, and the network-interface daemon controls communications. The statistical database records user and system statistics, the knowledge base consists of a representation of the current fact base and rule base, and the user interface provides communication between MIDAS and a SSO.

An expert system utilizes a forward chaining algorithm with four tiers (generations) of rules. The firing of some combination of rules in one tier can cause the firing of a rule in the next tier. The higher the tier, the more specific the rules become in regards to the possibility of attacks.

MIDAS keeps user and system-wide statistical profiles that record the aggregation of monitored system activity. The user's (system's) current session profile is compared to the historical profile to determine whether or not the current activity is outside two standard deviations.

2.7.3. System Operation

The logical structure of the MIDAS system revolves around the rules (heuristics) contained in the rule base. There are currently three different types of rules that MIDAS employs to review audit data.

1) **Immediate Attack.** These rules examine a small number of data items without using any kind of statistical information. They are intended to find only those auditable events that are, by themselves, abnormal enough to raise suspicion.

2) **User Anomaly.** These rules use statistical profiles to detect when a user's behavior profile deviates from previously-observed behavior patterns. User profiles are updated at the end of a user's session if the behavior has changed significantly, and are maintained for each user throughout the life of the account.

3) **System State.** These rules are similar to the user anomaly rules, but depict what is normal for the entire system, rather than for single users.

2.8. Wisdom and Sense

2.8.1. Overview

Wisdom and Sense (W&S) is an anomaly detection system developed at the Los Alamos National Laboratory [15] which operates on a Unix (IBM RT/PC) platform and analyzes audit trails from VAX/VMS hosts. It is an anomaly detection system which seeks to identify system usage patterns which are different from historical norms. It can process audit trail records in real time, although it is hampered by the fact that the operating system may delay writing the audit records.

The objectives of W&S are to detect intrusions, malicious or erroneous behavior by users, Trojan horses, and viruses. The system is based on the presumption that such behavior is anomalous and could be detected by comparing audit data produced by them with that of routine operation.

2.8.2. System Organization

W&S is a statistical, rule-based system. One of its major features is that it derives its own rule base from audit data. It receives historical audit data from the operating system and processes it into rules. These are formed into a forest (i.e., a set of trees). The rules are human-readable, and thus the rule base may be supplemented or modified by a human expert to correct deficiencies and inconsistencies. The rules define patterns of normal behavior in the system. A W&S rule base may contain between 10^4 and 10^6 rules, which take 6 to 8 bytes each, and can be searched in about 50ms.

The system views the universe as a collection of events, each represented by an audit record. Audit log records contain data about the execution of individual processes. Each record consists of a number of fields that contain information such as the invoker (user), the name of the process, it's privileges, and system resources utilized.

Data is viewed primarily as categorical, i.e., any field in a record can take one of a number of values. Categorical data are represented as character strings. Continuous data, such as CPU time, are mapped into a set of closed ranges, and then treated as categorical data.

2.8.3. System Operation

1) **Rules.** Rules consist of a Left Hand Side (LHS), which specifies the conditions under which the rule applies, and a Right Hand Side (RHS) (also

referred to as the rule's restriction), which defines what is considered normal under these conditions. The absence of a rule means that everything is considered normal.

The LHS could consist of field values or value ranges, values computed from a series of records (e.g. mean time between events), or subroutines returning a Boolean value. A given rule fires only if an audit record has fields whose values match the LHS and any subroutines in the LHS return true.

The RHS may take the form of a list of acceptable categorical values for a record field, a list of acceptable ranges of a continuous field, and a list of user-defined functions. Each rule has a grade, which is a measure of its accuracy. Rules which are more specific, or which represent frequently-occurring patterns with less variability, are given better grades.

2) **Constructing the Rule Base.** The historical data is first condensed, and then processed through the rule base generator, which builds the forest of rule trees. At each level, the rule base consists of nodes designating fields, and nodes designating acceptable values of each field. The rules are generated by repeatedly sorting the data and examining the frequency of field values. The tree is pruned as it is being built by using a number of pruning rules to limit its size.

3) **Audit Data Analysis.** The "Sense" part of W&S analyzes an activity file using the rule forest. It looks at a record, finds the applicable rules, and computes a *figure of merit* (FOM) for each field and each transaction. A transaction's FOM is the normalized sum of the grades of failed rules.

A number of transactions may be grouped to form a *thread*. Each thread belongs to a *thread class* which is defined by values of specific audit record fields. Some thread classes used are: each user/terminal combination, each program/user combination, and each privilege level. A set of operations may be defined for each thread class and carried out whenever a record in the class is processed. An FOM is computed for each thread as a time-decayed sum of the FOM's of its transactions. A transaction, or a thread, is considered anomalous if its FOM is above a pre-defined threshold.

The Sense module also provides an interactive interface to the configuration settings, rule base maintenance routines, and analysis tools. W&S offers several aids to the task of explaining the meaning and cause of anomalous events. It has undergone operational testing and has detected interesting anomalies even in data originally thought to be free of such events.

3. Network Security Monitor

3.1. Overview

The *Network Security Monitor* (NSM) is different from the intrusion detection systems discussed in Section 2 in that it does not analyze audit trails to detect intrusive behavior. The NSM, as the name implies, analyzes the traffic on a broadcast local area network to detect intrusive behavior. The reasons for this departure from the standard intrusion detection methods are outlined as follows.

First, although most IDSs are designed with the goal of supporting a number of different operating system platforms, all present audit-trail-based IDSs have only been used on a single operating system at any one time. These systems are usually designed to transform an audit log into a proprietary format used by the IDS [6, 10, 11]. In theory, audit logs from different operating systems need only to be transformed into this proprietary form for the IDS to perform its analysis. However, no results of an IDS successfully supporting multiple operating systems have been reported.

On the other hand, standard network protocols exist (e.g., TCP/IP and UDP/IP) which most major operating systems support and use. By using these network standards, the NSM can monitor a heterogeneous set of hosts and operating systems simultaneously.

Second, audit trails are often not available in a timely fashion. Some IDSs are designed to perform their analysis on a separate host, so the audit logs must be transferred from the source host to the second host monitor [11]. Furthermore, the operating system can often delay the writing of audit logs by several minutes [15]. The broadcast nature of local area networks, however, gives the NSM instant access to all data as soon as this data is transmitted on the network. It is then possible to immediately start the attack detection process.

Third, the audit trails are often vulnerable. In some past incidents, the intruders have turned off audit daemons or modified the audit trail. This action can either prevent the detection of the intrusion, or it can remove the capability to perform accountability (who turned off the audit daemons?) and damage control (what was seen, modified, or destroyed?) The NSM, on the other hand, passively listens to the network, and is therefore logically protected from subversion. Since the NSM is invisible to the intruder, it cannot be turned off (assuming it is physically secured), and the data it collects cannot be modified.

Fourth, the collection of audit trails degrades the performance of a machine being monitored. Unless audit trails are being used for accounting purposes, system administrators often turn off auditing. If analysis of these audit logs is also to be performed on the host, added degradation will occur. If the audit logs are transferred across a network or communication channel to a separate host for analysis, the loss of network bandwidth, as well as the loss of timeliness of the data will occur. In many environments, the degradation of monitored hosts or the loss of network bandwidth may discourage administrators from using such an IDS. The alternative, viz. the NSM architecture, does not degrade the performance of the hosts being monitored. The monitored hosts are not aware of the NSM, so the effectiveness of the NSM is not dependent on the system administrator's configuration of the monitored hosts.

And, finally, many of the more seriously documented cases of computer intrusions have utilized a network at some point during the intrusion, i.e., the intruder was physically separated from the target. With the continued proliferation of networks and interconnectivity, the use of networks in attacks will only increase. Furthermore, the network itself, being an important component of a computing environment, can be the object of an attack. The NSM can take advantage of the increase of network usage to protect the hosts attached to the networks. It can monitor attacks launched against the network itself, an attack that host based audit trail analyzers would probably miss.

3.2. The NSM Model

The NSM models the network and hosts being monitored in the hierarchically-structured Interconnected Computing Environment Model (ICEM). The ICEM is composed of six layers, the lowest being the bit streams on the network, and the highest being a representation for the state of the entire networked system.

The bottom-most, or first, layer is the *packet layer*. This layer accepts as input a *bit stream* from a broadcast local area network, viz. an Ethernet. The bit stream is divided up into complete Ethernet packets, and a time stamp is attached to the packet. This *time-augmented packet* is then passed up to the second layer.

The next layer, called the *thread layer*, accepts as input the time-augmented packets from the packet layer. These packets are then correlated into unidirectional data streams. Each stream consists of the data (with the different layers of protocol headers removed) being transferred from one host to another host by a particular protocol (TCP/IP or UDP/IP), through a unique set (for the particular set of hosts and protocol) of ports. This stream of data, which is called a thread, is mapped into a *thread vector*. All the thread vectors are passed up to the third layer.

The *connection layer*, which is the third layer, accepts as input the thread vectors generated by the thread layer. Each thread vector is paired, if possible, to another thread vector to represent a bidirectional stream of data (i.e., a host-to-host connection). These pairs of thread vectors are represented by a connection vector generated by the combination of the individual thread vectors. Each connection vector will be analyzed, and a reduced representation, a *reduced connection vector*, is passed up to the fourth layer.

Layer 4 is the *host layer* which accepts as input the reduced connection vectors generated by the connection layer. The connection vectors are used to build *host vectors*. Each host vector represents the network activities of a single host. These host vectors are passed up to the fifth layer.

The *connected network* layer is the next layer in the ICEM hierarchy. It accepts as input the host vectors generated by the host layer. The host vectors are transformed into a graph G by treating the Data_path_tuples of the host vectors as an adjacency list. If G(host1,host2,serv1) is not empty, then there is a connection, or path, from host1 to host2 by service serv1. The value for location G(host1,host2,serv1) is non empty if the host vector for host1 has (host2,serv1) in it's Data_path_tuples. This layer can build the connected sub-graphs of G, called a *connected network vector*, and compare these sub-graphs against historical connected sub-graphs. This layer can also accept questions from the user about the graph. For example, the user may ask if there is some path between two hosts - through any number of intermediate hosts - by a specific service. This set of connected network vectors is passed up to the sixth and final layer.

The top-most layer, called the *system layer*, accepts as input the set of connected network vectors from the connected network layer. The set of connected network vectors are used to build a single *system vector* representing the behavior of the entire system.

3.3. Detecting Intrusive Behavior

The traffic on the network is analyzed by a simple expert system. The types of inputs to the expert system are described below.

The current traffic cast into the ICEM vectors as discussed in the previous subsection is the first type of input. Currently, only the connection vectors and the host vectors are used. The components for these vectors are presented in Tables I and II.

The profiles of expected traffic behavior are the second type input. The profiles consist of expected data paths (viz. which systems are expected to establish communication paths to which other systems, and by which service?) and service profiles (viz. what is a typical *telnet*, *mail*, *finger*, etc., expected to look like?) Combining profiles and current network traffic gives the NSM the ability to detect anomalous behavior on the network.

The knowledge about capabilities of each of the network services is the third type of input (e.g., *telnet* provides the user with more capability than *ftp* does).

The level of authentication required for each of the services is the fourth type of input (e.g., *finger* requires no authentication, *mail* requests authentication but does not verify it, and *telnet* requires verified authentication).

The level of security for each of the machines is the fifth type of input. This can be based on the NCSC rating of machines, history of past abuses on the different machines, the rating received after running system evaluation software such as SPI or COPS, or simply which machines the security officer has some control over and which machines the security officer has no control over (e.g., a host from outside the local area network).

And the signatures of past attacks is the sixth type of input. Examples include seeing the vertical bar symbol (i.e., |, a Unix "pipe" symbol) in the receiver address for *mail*, or *finger* connections where the initiating host sends more than 512 bytes to the receiving host.

The data from these sources is used to identify the likelihood that a particular connection represents intrusive behavior, or if a host has been compromised. The security_state, or suspicion level, of a particular connection is a function of the abnormality of the connection, the security level of the service being used for the connection, the direction of the connection security level, and the matched signatures of attacks in the data stream for that connection.

The abnormality of a connection is based on the probability of that particular connection occurring and the behavior of the connection itself. If a connection from host A to host B by service C is rare, then the abnormality of that connection is high. Furthermore, if the profile of that connection compared to a typical connection by the same type of service is unusual (e.g., the number of packets or bytes is unusually high for a *mail* connection), the abnormality of that connection is high.

The security level of the service is based on the capabilities of that service and the authentication required by that service. The *tftp* service, for example, has great capabilities with no authentication, so the security level for *tftp* is high. The *telnet* service, on the other hand, also has great capabilities, but it also requires strong authentication. Therefore, the security level for *telnet* is lower than that of *tftp*.

The direction of connection security level is based on the security levels of the two machines involved and the which host initiated the connection. If an low security host connects to, or attempts to connect to a high security host, the direction of connection security level of that connection is high. On the other hand, if a high security host connects to an insecure host, the direction of connection security level is low.

The matched strings consists of the vectors **Initiator_X** and **Receiver_X**. Thus it is simply a list of counts for the number of times each string being searched for in the data is matched.

The connection vectors are essentially treated as records in a database, and presentation of the information may be made as simple requests into the database. The default presentation format sorts the connection by suspicion level and presents the sorted list from highest suspicion level to the lowest. Presentations can also be made by specifying time windows for connection, connections from a specific host, connections with a particular string matched, etc.

The security_state, or suspicion level, of a host is simply the maximum security_state of its connection vectors over particular window of time. The host vectors are also treated as records into a database, and they may be presented in a similar fashion as the connection vectors.

4. Distributed Intrusion Detection System (DIDS)

4.1. Introduction

Our previous work concentrated on the development of an intrusion-detection model and a prototype implementation of a network security monitor (NSM) for a broadcast local area network environment [5]. The NSM (adaptively) develops profiles of usage of network resources and then compares current usage patterns with the historical profile to determine possible security violations. The goal of our proposed research is to extend our network intrusion-detection concept from the LAN environment to arbitrarily wider areas with the network topology being arbitrary as well.

The generalized distributed environment is heterogeneous, i.e., the network nodes can be hosts or servers from different vendors, or some of them could be LAN monitors, like our NSM, as well. The proposed architecture for this distributed intrusion-detection system consists of three major components. First, there is a host monitor on each host that is required to be monitored by the system. This monitor is a collection of processes running in background in the host. Second, each LAN segment has a LAN monitor, which operates just like a host monitor except that it analyzes LAN traffic. Finally, there is the DIDS director which is placed at a single secure location. The director receives reports from various host and LAN monitors, and by processing and correlating these reports, it is expected to detect intrusions.

4.2. Design Goals

DIDS is a proposed distributed intrusion detection system that is intended to enhance the effectiveness and efficiency of the SSO in monitoring a network of computers. DIDS is designed to discover attacks on, and misuse of, individual hosts as well as the network which connects them. It looks for attempts to subvert or avoid authentication as well as other methods of gaining unauthorized access to, or privileges on, the monitored computers. It also attempts to detect masqueraders and insiders committing a variety of offenses, e.g., espionage or data alteration. The system is based on both known and hypothesized abuses. It is designed to operate in near real time, providing for both general surveillance and focused investigation. The analysis portion of the system utilizes an inference network and incorporates learning algorithms so that it can deal with new forms of attacks and abuse as they develop. The inference network is also supported by an explanation facility which lets the operator examine the system's chain of reasoning. DIDS incorporates various ideas from a number of its predecessors. In addition to refining these approaches, DIDS provides a new dimension to intrusion detection by facilitating the correlation and analysis of data from multiple sources. The target environment will consist of a single physical segment of a local area network with approximately 10 hosts running at least 2 different C2-level[1] secure operating systems.

4.3. Overview

In DIDS, the monitoring and analysis functions are distributed among several components. These components include a *DIDS director*, a collection of *host monitors*, and at least one *LAN monitor*. The host and LAN monitors are primarily responsible for detecting single events and known attack signatures which have a high probability of being relevant to the security of a system; so they must constantly monitor their respective domains. The director is responsible for analyzing the events reported by the host and LAN monitors, and therefore will have access to the distributed audit data gathered by the various monitors. The director may then use these records in support of a directed investigation of a particular subject. The director communicates bidirectionally with the host and LAN monitors to facilitate the transfer of data and the processing of queries and responses. It is also able to correlate information obtained from the individual host and LAN monitors. The director also supports the user interface.

The host monitor is a process or collection of processes running in the background on each individual host, making the presence of the host monitor essentially transparent to the users of the host. The LAN monitor and the director are also processes running on single hosts on the network. Ideally, the director should have exclusive use of a host since it is responsible for the high-level analysis of the data, and it will require significant computing resources. Although the LAN monitor, the host monitor, and the director may physically reside on the same computer, they are logically independent (see Fig. 1).

The division of labor between the central and distributed components of DIDS is straight-forward. Correlation of the data from multiple sources must be done at the director. Platform specific transformations of information should clearly be done on the individually monitored hosts. However, there is a trade-off between doing some of the analysis on the distributed monitors and doing it all on the director. If more work is done on the distributed monitors, it will reduce the chance that the processing capability of the director will be a bottleneck, and it will also reduce the amount of data that must be sent across the network. On the other hand, increasing the amount of analysis that is done on the distributed monitors will obviously hinder the performance of the individual hosts.

4.4. Host Monitor

The major components of the host monitor deal with capturing and translating audit data, simple analysis of the translated audit data, and communication with the DIDS director.

The host monitor uses the native audit data provided by the host's operating system, so that the operating system itself is not modified. The host monitor captures each audit record from the operating system, which is then mapped into a common syntax called a *Host Audit Record* (HAR). This common syntax across operating systems provides the first level of abstraction from the native audit record format and improves portability and heterogeneity of the remaining portions of the host monitor. Redundant records are also eliminated at this point. This provides a significant reduction in the number of records that the higher levels of the host

[1] The DOD defines Class (C2) as controlled access protection, i.e., finely grained discretionary access control that makes users individually accountable for their actions (see [1]).

monitor needs to process (more than a 4:1 reduction has been observed in early tests).

The host monitor incorporates three levels of analysis performed on the HARs. At the lowest level, the host monitor scans each HAR for *notable events*. Notable events are transactions that may be of interest independent of their context (i.e., independent of previous HARs). Examples of notable events include any type of network activity, failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the host monitor looks for *sequences* of events which may be interesting. Known attack signatures or patterns of abuse are examples of sequences which would be of interest. We are developing a general purpose approximate pattern matcher which will be used for this purpose. Finally, the host monitor looks for anomalous behavior by tracking user behavior patterns, such as number of programs executed, number of files accessed, etc. As noted above, there are performance tradeoffs to be considered when deciding how much analysis should be done at the host monitor. When a host monitor notes an interesting event or pattern, it alerts the director and forwards the relevant information. The lists of notable events, the templates for pattern matching, and the metrics for anomaly detection are maintained separately to allow for easy modification.

The audit data for each host is, in general, stored locally. Since our target platform is a C2 or higher rated system the audit data is already being kept as part of the C2 specification [1] so this imposes no additional burden on the host. Moreover, unlike some systems, we do not use network bandwidth in transmitting all the audit data to the director. In addition to storing the native audit data, the host keeps a short history of the translated records produced by the host monitor. This history simplifies the process of providing information to the director. The number and duration of these records depends on both system resources and security needs.

Although the audit records provide a significant amount of information, there are still a few metrics that are, at present, best gotten by querying the host operating system directly (e.g., Unix's *ps*, Sun's *rusers*, etc.). To facilitate these queries, the host monitor provides an *agent* through which the director can ask for information about the status of the host. The host monitor also handles director queries of the stored HARs.

4.5. LAN Monitor

The DIDS LAN monitor is built on the same foundation as UC Davis' Network Security Monitor [5]. Since there is no native LAN audit trail, the LAN monitor is responsible for building its own. The LAN monitor sees every packet on its segment of the LAN. From these packets, the LAN monitor is able to construct higher level objects such as connections (logical circuits), and service requests. In particular, it audits host-to-host connections, services used, and volume of traffic. Like the host based monitor, the LAN monitor uses several levels of analysis to catch the most significant events, for example, sudden changes in network load, the use of security-related services, and network activities such as *rlogin*. As with the host monitor, the LAN monitor retains the audit data for analysis by the director. It also uses and maintains profiles of network behavior, which are updated periodically. Like the host monitor, the LAN monitor provides an agent for communications with the director. In addition to handling queries of the audit data from the director, this agent gives the director access to a number of network management tools, which are analogous to the host operating system services provided by the host monitor.

4.6. DIDS Director

The director consists of four major components: an interprocess *communications manager*, an *expert system*, a set of *tools*, and a *user interface*.

4.6.1. Communications Manager

The *communications manager* is responsible for the coordination of information and instructions between the director and each of the host and the LAN monitors. In particular, the communications manager interprets requests from the expert system and relays them to the agents of the host and the LAN monitors; it also buffers the corresponding replies and converts them into the form that is expected by the expert system. The communications manager also processes the SSO's requests for the various monitors, tools, and operating systems. For example, the SSO, through the user interface, might request a listing of all current logins on a particular host. That request would pass through the communications manager, which would form an appropriate call to the target OS. When a response is received from the host monitor, the communications manager would then return that response to the user interface. Network protocols are provided so that the details of the communications between the various components of DIDS are invisible to the individual components (hosts and LANs), as well as to the SSO.

4.6.2. Expert System

The *expert system* is responsible for evaluating and reporting the security state of the monitored network. It receives the reports from the host and the LAN monitors through the communications manager, and, based on these reports, it makes inferences about the security of users, hosts, processes, etc. The expert system reports its conclusions through the user interface. The expert system consists of three components: the *evidence*, which are facts about the world; the *inference rules*, which represent the knowledge which the expert uses in reasoning about the evidence; and the *inference mechanism*, which controls the process of reasoning.

The evidence takes the form of audit records residing on the individual hosts and on the LAN monitor. In addition, the expert system has access to databases of (relatively) static information regarding the hosts, LANs, and users. This information could include such things as: user SMITH is on vacation, or host EREBUS is out of service. In its local fact base, the expert system has access to its own intermediate hypotheses as well as the notable events which have been reported by the LAN and the host monitors.

Except for the information in the static databases, all of the facts are affected by the passage of time. The individual audit files are periodically purged of the oldest information. The intermediate hypotheses and notable events in the local fact base are also aged. Thus, the expert system recognizes the importance of temporal proximity without having to specify sequences of events, (see [14]).

The inference rules provide the ability to recognize a security incident. Each rule takes the general form of a conclusion, which is logically dependent on one or more antecedents. The individual rules are then linked together to form an inference network. That is, the antecedents of a rule are, in general, the conclusions of other rules. The rule base has a hierarchical structure based on a model which describes a security incident in terms of levels of abstraction from the evidence. Reasoning progresses up the hierarchy of abstractions using rules whose conclusions are at a higher level than their antecedents. The current version of the model has seven layers of abstraction. The model provides a framework for developing the rules themselves, as well as for providing the foundation for our argument that the rule base is comprehensive.

The principle issues related to the inferencing mechanism are handling uncertainty and training. There are few situations in which one can be completely sure whether a user's behavior represents an abuse. Therefore, a mechanism which attempts to evaluate security incidents must be capable of reasoning with uncertainty. To this end, rules have a *Rule Value* (RV) associated with them. This is similar to the certainty factor of EMYCIN [4]. The RV represents the degree to which the truth of the antecedents of a rule guarantees the truth of the conclusion. The RV is expressed as number between 0 (no faith) and 100 (absolute faith).

For each *intermediate hypothesis* (i.e., conclusion of some rule) the system maintains a *Hypothesis Value* (HV). The HV is expressed as a value between 0 (no support for the hypothesis) and 100 (the hypothesis is absolutely true). HVs automatically decrease over time as a perceived threat diminishes.

A simple algorithm is used for combining HVs and RVs. In applying rules whose antecedents are uncertain, the HV of the antecedent is multiplied by the RV of the rule to get an effective HV. The effective HV is then applied to the conclusion of the rule. In the case of rules with disjunctive antecedents, the maximum of the applicable HVs is used; in cases of conjunctive antecedents, the minimum of the applicable HVs is used; in the case in which multiple rules support a single hypothesis, the following formula is used:

$$HV_1 = HV_1 + \frac{(100 - HV_1) \times HV_2}{100}$$

where HV_1 is the HV of the hypothesis and HV_2 is the effective HV generated by the new application of a rule. This simple method avoids the special case calculations of MYCIN's certainty factors and the need for conditional probabilities as in the Bayesian approach [8].

While inductive learning has been successfully applied to the problem of recognizing anomalies [15], it does not apply to the problem of recognizing abuses. This is because there is no general criterion for recognizing an abuse as there is for recognizing an anomaly. On the other hand, it seems unlikely that one could specify a rule base using traditional knowledge engineering methods. It is particularly unlikely that this approach could provide useful certainty values. As an alternative to either of these approaches, a method based on negative reinforcement is proposed. Essentially, the expert system will be taught to recognize what is *not* an abuse. The initial rule base is intentionally broad, with all of the rules initially given high values. The untrained system is thus overly sensitive and reports many security incidents which are false alarms. A human trainer is required to decide which of the reported incidents are valid. When the trainer ascertains that a report is invalid (i.e., a false alarm), he/she initiates a process which automatically reduces the RV of each rule that was used in inferring the security incident. This approach is fundamentally different from the inductive learning method used by other IDSs in that it makes no assumptions about the content of the training data. What is required is that the training proceed until the system makes acceptably-few mistakes. To avoid having the system become too desensitized, the RV of all active rules uniformly and gradually increases over time. This has the effect of keeping the system tuned.

This training method has the advantage that a general system can be created which will tailor itself to a particular installation or class of installations. Also, as a side effect of this training method, we get a reasonable approximation of anomaly detection. Since all of the rules are initialized with high RVs, and the RVs are only lowered when a rule is used in drawing a false inference, those rules whose antecedents are seldom seen will continue to have high RVs after the initial training.

4.6.3. Tools

We anticipate that a growing set of tools will be used in conjunction with the intrusion detection functions of DIDS. Among these will include incident handling tools. Depending on the level of expertise of the SSO, these tools could include online explicit instructions, e.g., "who to call" lists, automatic response procedures, etc. In addition to the incident handling tools, the user interface can provide access to standard network administration tools; in particular, we are evaluating the SNMP tools currently available.

4.6.4. User Interface

The director's user interface serves both the intrusion detection and the incident management components of DIDS. It is designed to take advantage of the graphic capabilities of the target platform (a Sun SPARCstation 1). The user interface is hierarchical in its presentation of information, allowing the SSO to display as much, or as little, of the information available from DIDS as he/she wants. In its most compact form, the interface is reduced to a state meter displaying a measure of the security state of the network as evaluated by DIDS. Other options include a control panel for access to system management tools, displays of summary or detailed monitoring of the net, of hosts or of users, and a query capability for the distributed audit data. The user interface is designed to be extensible, so that new functionality can be added without undue effort.

5. Conclusion

This paper presented an architecture for a Distributed Intrusion Detection System (DIDS). The target environment for DIDS is a heterogeneous network of computers that may consist of different hosts, servers, etc. The DIDS architecture consists of a collection of host monitors, each of which monitors a single host computer; one or more LAN monitors, each of which monitors traffic on its corresponding LAN segment; and the DIDS director, which receives reports from the various monitors, correlates the data, and makes inferences regarding the security state of the system.

We believe that DIDS will be able to detect the same kind of single host intrusions that are flagged by other intrusion detection systems, such as IDES [6], Wisdom & Sense [15], and MIDAS [10]. DIDS should also be able to (1) detect attacks on the network itself, (2) detect attacks involving multiple hosts, (3) track tagged objects, including users and sensitive files, as they move around the network, (4) detect, via erroneous or misleading reports, situations where a host might be taken over by an attacker, and (5) monitor the activity of any networked system that doesn't have a host monitor, yet generates LAN activity, such as a PC.

References

1. *Department of Defense Trusted Computer System Evaluation Criteria*, Dept. of Defense, National Computer Security Center, DOD 5200.28-STD, Dec. 1985.
2. D.E. Denning, "An Intrusion-Detection Model," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 2, pp. 222-232, Feb. 1987.
3. C. Dowell and P. Ramstedt, "The COMPUTERWATCH Data Reduction Tool," *Proc. 13th National Computer Security Conference*, pp. 99-108, Washington, D.C., Oct. 1990.
4. R. Forsyth, ed., *Expert Systems, Principles and Case Studies*, Chapman & Hall, London, 1989.
5. L.T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc. 1990 Symposium on Research in Security and Privacy*, pp. 296-304, Oakland, CA, May 1990.
6. T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P.G. Neumann, and C. Jalali, "IDES: A Progress Report," *Proc. Sixth Annual Computer Security Applications Conference*, Tucson, AZ, Dec. 1990.
7. T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, H.S. Javitz, A. Valdes, and P.G. Neumann, "A Real-Time Intrusion-Detection Expert System (IDES)," Interim Progress Report, Project 6784, SRI International, May 1990.
8. K.C. Ng and B. Abramson, "Uncertainty Management in Expert Systems," *IEEE Expert*, Apr. 1990.
9. K.L. Pon and A.A. Garcia, "A Computer Security Monitor (DRAFT)," Technical Report, Lawrence Livermore National Laboratory, March 1989.
10. M.M. Sebring, E. Shellhouse, M.E. Hanna, and R.A. Whitehurst, "Expert Systems in Intrusion Detection: A Case Study," *Proc. 11th National Computer Security Conference*, pp. 74-81, Oct.

1988.

11. S.E. Smaha, "Haystack: An Intrusion Detection System," *Proc. IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, Dec. 1988.

12. W.T. Tener, "Discovery: an expert system in the commercial data security environment," *Security and Protection in Information Systems: Proc. Fourth IFIP TC11 International Conference on Computer Security*, North-Holland, Dec. 1986.

13. W.T. Tener, "AI & 4GL: Automated Detection and Investigation Tools," *Computer Security in the Age of Information: Proc. Fifth IFIP International Conference on Computer Security*, North-Holland, May 1988.

14. H.S. Teng, K. Chen, and S.C-Y Lu, *Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns.*

15. H.S. Vaccaro and G.E. Liepins, "Detection of Anomalous Computer Session Activity," *Proc. Symposium on Research in Security and Privacy*, pp. 280-289, Oakland, CA, May 1989.

16. J.R. Winkler and W.J. Page, "Intrusion and Anomaly Detection in Trusted Systems," *Proc. 5th Annual IEEE Computer Security Applications Conference*, pp. 39-45, Dec. 1989.

17. J.R. Winkler, "A Unix Prototype for Intrusion and Anomaly Detection in Secure Networks," *Proc. 13th National Computer Security Conference*, pp. 115-124, Washington, D.C., Oct. 1990.

Table I. CONNECTION VECTOR

Component	Description
Connection_ID	Unique integer used to reference this particular connection.
Initiator_address	The internet address of the host which initiated the connection.
Receiver_address	The internet address of the host to which the connection was made.
Service	An integer used to identify the particular service (i.e., *telnet* or *mail*) used for this connection.
Start_time	The time stamp on the first packet received for this connection.
Delta_time	The difference between the time stamp of the most recent packet for this connection and the Start_time.
Connection_state	The state of the connection. States for a connection include information such as: NEW_CONNECTION, CONNECTION_IN_PROGRESS, and CONNECTION_CLOSED.
Security_state	The current evaluation of the security state of this connection.
Initiator_pkts	The number of packets the host which initiated the connection has placed on the network.
Initiator_bytes	The number of bytes, excluding protocol headers, contained in the packets.
Receiver_pkts	The number of packets the host which received the connection has placed on the network.
Receiver_bytes	The number of bytes, excluding protocol headers contained in the packets.
Dimension	The dimension of the Initiator_X and the Receiver_X vectors. This value is the number of strings patterns being looked for in the data.
Initiator_X	A vector representing the number of strings matched in Initiator_bytes.
Receiver_X	A vector representing the number of strings matched in Receiver_bytes.

Table II. HOST VECTOR

Component	Description
Host_ID	Unique integer used to reference this particular host.
Host_address	The internet address of this host.
Host_state	The state of the host. States include: ACTIVE, NOT_ACTIVE.
Security_state	The current evaluation of the security of this particular host.
Data_path_number	The number of data paths the currently connected to this host. It may be considered the number of arcs from a node in a graph.
Data_path_tuples	A list of four-tuple representing a data path from or to the host. The tuple consists of: Other_host_address, Service_ID, Initiator_tag, and Security_state (of the data path).

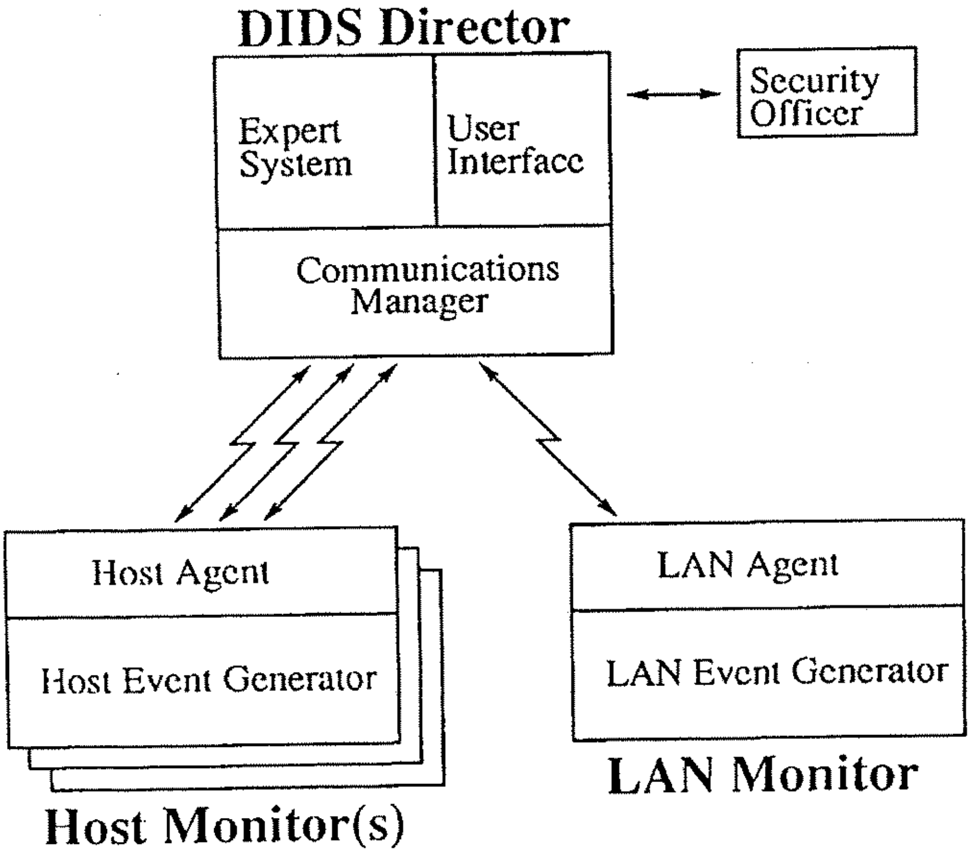


Fig. 1: Architecture of the Distributed Intrusion Detection System

C

DIDS (Distributed Intrusion Detection System) – Motivation, Architecture, and An Early Prototype

Steven R. Snapp[1], *James Brentano*[2], *Gihan V. Dias*, *Terrance L. Goan*,
L. Todd Heberlein, *Che-Lin Ho*, *Karl N. Levitt*, *Biswanath Mukherjee*, *Stephen E. Smaha*[1],
Tim Grance[3], *Daniel M. Teal*[3], *and Doug Mansur*[4]

Computer Security Laboratory
Division of Computer Science
University of California, Davis
Davis, California 95616

ABSTRACT

Intrusion detection is the problem of identifying unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators. The proliferation of heterogeneous computer networks provides additional implications for the intrusion detection problem. Namely, the increased connectivity of computer systems gives greater access to outsiders, and makes it easier for intruders to avoid detection. IDS's are based on the belief that an intruder's behavior will be noticeably different from that of a legitimate user. We are designing and implementing a prototype Distributed Intrusion Detection System (DIDS) that combines distributed monitoring and data reduction (through individual host and LAN monitors) with centralized data analysis (through the DIDS director) to monitor a heterogeneous network of computers. This approach is unique among current IDS's. A main problem considered in this paper is the Network-user Identification problem, which is concerned with tracking a user moving across the network, possibly with a new user-id on each computer. Initial system prototypes have provided quite favorable results on this problem and the detection of attacks on a network. This paper provides an overview of the motivation behind DIDS, the system architecture and capabilities, and a discussion of the early prototype.

1. Introduction

Intrusion detection is defined to be the problem of identifying individuals who are using a computer system without authorization (i.e., *crackers*) and those who have legitimate access to the system but are exceeding their privileges (i.e., the *insider threat*). Work is being done elsewhere on Intrusion Detection Systems (IDS's) for a single host [8, 10, 11] and for several hosts connected by a network [6, 7, 12]. Our own earlier work on the Network Security Monitor (NSM) concentrated on monitoring a broadcast Local Area Network (LAN) [3].

The proliferation of heterogeneous computer networks has serious implications for the intrusion detection problem. Foremost among these implications is the increased opportunity for unauthorized access that is provided by the network's connectivity. This problem is exacerbated when dial-up or internetwork access is allowed, as well as when unmonitored hosts (viz. hosts without audit trails) are present. The use of distributed rather than centralized computing resources also implies reduced control over those resources. Moreover, multiple independent computers are likely to generate more audit data than a single computer, and this audit data is dispersed among the various systems. Clearly, not all of the audit data can be forwarded to a single IDS for analysis; some analysis must be accomplished locally.

[1] Haystack Laboratories, Inc., 8920 Business Park Dr, Suite 270, Austin, TX 78759
[2] Pacific Gas and Electric Company, 77 Beale St, Room 1871B, San Francisco, CA 94106
[3] United States Air Force Cryptologic Support Center, San Antonio, TX 78243
[4] Lawrence Livermore National Labs, Livermore, CA 94550

This paper describes a prototype Distributed Intrusion Detection System (DIDS) which generalizes the target environment in order to monitor multiple hosts connected via a network as well as the network itself. The DIDS components include the DIDS director, a single host monitor per host, and a single LAN monitor for each LAN segment of the monitored network. The information gathered by these distributed components is transported to, and analyzed at, a central location (viz. an expert system, which is a sub-component of the director), thus providing the capability to aggregate information from different sources. We can cope with any audit trail format as long as the events of interest are provided.

DIDS is designed to operate in a heterogeneous environment composed of C2 [1] or higher rated computers. The current target environment consists of several hosts connected by a broadcast LAN segment (presently an Ethernet, see Fig. 1). The use of C2-rated systems implies a consistency in the content of the system audit trails. This allows us to develop standard representations into which we can map audit data from UNIX, VMS, or any other system with C2 auditing capabilities. The C2 rating also guarantees, as part of the Trusted Computing Base (TCB), the security and integrity of the host's audit records. Although the hosts must comply with the C2 specifications in order to be monitored directly, the network related activity of non-compliant hosts can be monitored via the LAN monitor. 'Since all attacks that utilize the network for system access will pass through the LAN segment, the LAN monitor will be able to monitor all of this traffic.

Section 2 motivates our work by describing the type of behavior which DIDS is intended to detect. In Section 3 we present an overview of the DIDS architecture. In Section 4 we formulate the concept of the network-user identification (NID), an identifier for a network-wide user, and describe its use in distributed intrusion detection. Sections 5 and 6 deal with the host and LAN monitors, respectively, while Section 7 discusses the expert system and its processing mechanisms based on the NID. Section 8 provides some concluding remarks.

2. Scenarios

The detection of certain attacks against a networked system of computers requires information from multiple sources. A simple example of such an attack is the so-called *doorknob* attack. In a doorknob attack the intruder's goal is to discover, and gain access to, insufficiently-protected hosts on a system. The intruder generally tries a few common account and password combinations on each of a number of computers. These simple attacks can be remarkably successful [4]. As a case in point, UC Davis' NSM recently observed an attacker of this type gaining super-user access to an external computer which did not require a password for the super-user account. In this case, the intruder used *telnet* to make the connection from a university computer system, and then repeatedly tried to gain access to several different computers at the external site. In cases like these, the intruder tries only a few logins on each machine (usually with different account names), which means that an IDS on each host may not flag the attack. Even if the behavior is recognized as an attack on the individual host, current IDS's are generally unable to correlate reports from multiple hosts; thus they cannot recognize the *doorknob* attack as such. Because DIDS aggregates and correlates data from multiple hosts and the network, it is in a position to recognize the doorknob attack by detecting the pattern of repeated failed logins even though there may be too few on a single host to alert that host's monitor.

In another incident, our NSM recently observed an intruder gaining access to a computer using a guest account which did not require a password. Once the attacker had access to the system, he exhibited behavior which would have alerted most existing IDS's (e.g., changing passwords and failed events). In an incident such as this, DIDS would not only report the attack, but may also be able to identify the source of the attack. That is, while most IDS's would report the occurrence of an incident involving user "guest" on the target machine, DIDS would also report that user "guest" was really, for example, user "smith" on the source machine, assuming that the source machine was in the monitored domain. It may also be possible to go even further back and identify all of the different user accounts in the "chain" to find the initial launching point of the attack.

Another possible scenario is what we call *network browsing*. This occurs when a (network) user is looking through a number of files on several different computers within a short period of time. The browsing activity level on any single host may not be sufficiently high enough to raise any alarm by itself. However, the network-wide, aggregated browsing activity level may be high enough to raise suspicion on this user. Network browsing can be detected as follows. Each host monitor will report that a particular user is browsing on that system, even if the corresponding degree of browsing is small. The expert system can then aggregate such information from multiple hosts to determine that all of the browsing activity corresponds to the same network user. This scenario presents a key challenge for DIDS: the tradeoff between sending all audit records to the director versus missing attacks because thresholds on each host are not exceeded.

In addition to the specific scenarios outlined above, there are a number of general ways that an intruder can use the connectivity of the network to hide his trail and to enhance his effectiveness. Some of the attack configurations which have been hypothesized include *chain* and *parallel* attacks [2]. DIDS combats these inherent vulnerabilities of the network by using the very same connectivity to help track and detect the intruder. Note that DIDS should be at least as effective as host-based IDS's (if we implement all of their functionality in the DIDS host monitor), and at least as effective as the stand-alone NSM.

3. DIDS Architecture

The DIDS architecture combines distributed monitoring and data reduction with centralized data analysis. This approach is unique among current IDS's. The components of DIDS are the *DIDS director*, a single *host monitor* per host, and a single *LAN monitor* for each broadcast LAN segment in the monitored network. DIDS can potentially handle hosts without monitors since the LAN monitor can report on the network activities of such hosts. The host and LAN monitors are primarily responsible for the collection of evidence of unauthorized or suspicious activity, while the DIDS director is primarily responsible for its evaluation. Reports are sent independently and asynchronously from the host and LAN monitors to the DIDS director through a communications infrastructure (Fig. 2). High level communication protocols between the components are based on the ISO Common Management Information Protocol (CMIP) recommendations, allowing for future inclusion of CMIP management tools as they become useful. The architecture also provides for bidirectional communication between the DIDS director and any monitor in the configuration. This communication consists primarily of notable events and anomaly reports from the monitors. The director can also make requests for more detailed information from the distributed monitors via a "GET" directive, and issue commands to have the distributed monitors modify their monitoring capabilities via a "SET" directive. A large amount of low level filtering and some analysis is performed by the host monitor to minimize the use of network bandwidth in passing evidence to the director.

The host monitor consists of a *host event generator* (HEG) and a *host agent*. The HEG collects and analyzes audit records from the host's operating system. The audit records are scanned for *notable events*, which are transactions that are of interest independent of any other records. These include, among others, failed events, user authentications, changes to the security state of the system, and any network access such as *rlogin* and *rsh*. These notable events are then sent to the director for further analysis. In enhancements under development, the HEG will also track user sessions and report anomalous behavior aggregated over time through user/group profiles and the integration of Haystack [10] into DIDS. The host agent handles all communications between the host monitor and the DIDS director.

Like the host monitor, the LAN monitor consists of a *LAN event generator* (LEG) and a *LAN agent*. The LEG is currently a subset of UC Davis' NSM [3]. Its main responsibility is to observe all of the traffic on its segment of the LAN to monitor host-to-host connections, services used, and volume of traffic. The LAN monitor reports on such network activity as *rlogin* and *telnet* connections, the use of security-related services, and changes in network traffic patterns.

The DIDS director consists of three major components that are all located on the same dedicated workstation. Because the components are logically independent processes, they could be distributed as well. The *communications manager* is responsible for the transfer of data between the director and each of the host and the LAN monitors. It accepts the notable event records from each of the host and LAN monitors and sends them to the *expert system*. On behalf of the expert system or user interface, it is also able to send requests to the host and LAN monitors for more information regarding a particular subject. The expert system is responsible for evaluating and reporting on the security state of the monitored system. It receives the reports from the host and the LAN monitors, and, based on these reports, it makes inferences about the security of each individual host, as well as the system as a whole. The expert system is a rule-based system with simple learning capabilities. The director's *user interface* allows the System Security Officer (SSO) interactive access to the entire system. The SSO is able to watch activities on each host, watch network traffic (by setting "wire-taps"), and request more specific types of information from the monitors.

We anticipate that a growing set of tools, including incident-handling tools and network-management tools, will be used in conjunction with the intrusion-detection functions of DIDS. This will give the SSO the ability to actively respond to attacks against the system in real-time. Incident-handling tools may consist of possible courses of action to take against an attacker, such as cutting off network access, a directed investigation of a particular user, removal of system access, etc. Network-management tools that are able to perform network mapping would also be useful.

4. The Network-user Identification (NID)

One of the more interesting challenges for intrusion detection in a networked environment is to track users and objects (e.g., files) as they move across the network. For example, an intruder may use several different accounts on different machines during the course of an attack. Correlating data from several independent sources, including the network itself, can aid in recognizing this type of behavior and tracking an intruder to their source. In a networked environment, an intruder may often choose to employ the interconnectivity of the computers to hide his true identity and location. It may be that a single intruder uses multiple accounts to launch an attack, and that the behavior can be recognized as suspicious only if one knows that all of the activity emanates from a single source. For example, it is not particularly noteworthy if a user inquires about who is using a particular computer (e.g., using the UNIX *who* or *finger* command). However, it may be indicative of an attack if a user inquires about who is using each of the computers on a LAN and then subsequently logs into one of the hosts. Detecting this type of behavior requires attributing multiple sessions, perhaps with different account names, to a single source.

This problem is unique to the network environment and has not been dealt with before in this context. Our solution to the multiple user identity problem is to create a *network-user identification* (NID) the first time a user enters the monitored environment, and then to apply that NID to any further instances of the user. All evidence about the behavior of any instance of the user is then accountable to the single NID. In particular, we must be able to determine that "smith@host1" is the same user as "jones@host2", if in fact they are. Since the network-user identification problem involves the collection and evaluation of data from both the host and LAN monitors, examining it is a useful method to understand the operation of DIDS. In the following subsections we examine each of the components of DIDS in the context of the creation and use of the NID.

5. The Host Monitor

The host monitor is currently installed on Sun SPARCstations running SunOS 4.0.x with the Sun C2 security package [9]. Through the C2 security package, the operating system produces audit records for virtually every transaction on the system. These transactions include file accesses, system calls, process executions, and logins. The contents of the Sun C2 audit record are: record type, record event, time, real user ID, audit user ID, effective user ID, real group ID, process ID, error code, return value, and label.

The host monitor (Fig. 3) examines each audit record to determine if it should be forwarded to the expert system for further evaluation. Certain critical audit records are always passed directly to the expert system (i.e., *notable events*); others are processed locally by the host monitor (i.e., *profiles* and attack *signatures*, which are sequences of noteworthy events which indicate the symptoms of attacks) and only summary reports are sent to the expert system. Thus, one of the design objectives is to push as much of the processing operations down to the low-level monitors as possible. In order to do this, the HEG creates a more abstract object called an *event*. The event includes any significant data provided by the original audit record plus two new fields: the *action* and the *domain*. The action and domain are abstractions which are used to minimize operating system dependencies at higher levels. Actions characterize the dynamic aspect of the audit records. Domains characterize the objects of the audit records. In most cases, the objects are files or devices and their domain is determined by the characteristics of the object or its location in the file system. Since processes can also be objects of an audit record, they are also assigned to domains, in this case by their function.

The actions are: session_start, session_end, read (a file or device), write (a file or device), execute (a process), terminate (a process), create (a file or (virtual) device), delete (a file or (virtual) device), move (rename a file or device), change_rights, and change_user_id. The domains are: tagged, authentication, audit, network, system, sys_info, user_info, utility, owned, and not_owned.

The domains are prioritized so that an object is assigned to the first applicable domain. *Tagged* objects are ones which are thought a priori to be particularly interesting in terms of detecting intrusions. Any file, device, or process can be tagged (e.g., */etc/passwd*). *Authentication* objects are the processes and files which are used to provide access control on the system (e.g., the password file). Similarly, *audit* objects relate to the accounting and security auditing processes and files. *Network* objects are the processes and files not covered in the previous domains which relate to the use of the network. *System* objects are primarily those which are concerned with the execution of the operating system itself, again exclusive of those objects already assigned to previously considered domains. *Sys_info* and *user_info* objects provide information about the system and about the users of the system, respectively. The *utility* objects are the bulk of the programs run by the users (e.g., compilers and editors). In general, the execution of an object in the utility domain is not interesting (except when the use is excessive), but the creation or modification of one is. *Owned* objects are relative to the user. *Not_owned* objects are, by exclusion, every object not assigned to a previous domain. They are also relative to a user; thus, files in the owned domain relative to "smith" are in the not_owned domain relative to "jones".

All possible transactions fall into one of a finite number of events formed by the cross product of the actions and the domains, and each event may also succeed or fail. Note that no distinction is made between files, directories or devices, and that all of these are treated simply as objects. Not every action is applicable to every object; for example, the *terminate* action is applicable only to processes. The choice of these domains and actions is somewhat arbitrary in that one could easily suggest both finer and coarser grained partitions. However, they capture most of the interesting behavior for intrusion detection and correspond reasonably well with what other researchers in this field have found to be of interest [5, 10]. By mapping an infinite number of transactions to a finite number of events, we not only remove operating system dependencies, but also restrict the number of permutations that the expert system will have to deal with. The concept of the domain is one of the keys to detecting abuses. Using the domain allows us to make assertions about the nature of a user's behavior in a straightforward and systematic way. Although we lose some details provided by the raw audit information, that is more than made up for by the increase in portability, speed, simplicity, and generality.

An event reported by a host monitor is called a host audit record (har). The record syntax is: har(Monitor-ID, Host-ID, Audit-UID, Real-UID, Effective-UID, Time, Domain, Action, Transaction, Object, Parent Process, PID, Return Value, Error Code).

Of all the possible events, only a subset are forwarded to the expert system. For the creation and application of the NID, it is the events which relate to the creation of user sessions or to a change in an account that are important. These include all the events with *session_start* actions, as well as ones with an *execute* action applied to the *network* domain. These latter events capture such transactions as executing the *rlogin*, *telnet*, *rsh*, and *rexec* UNIX programs. The HEG consults external tables, which are built by hand, to determine which events should be forwarded to the expert system. Because they relate to events rather than to the audit records themselves, the tables and the modules of the HEG which use them are portable across operating systems. The only portion of the HEG which is operating system dependent is the module which creates the events.

6. The LAN Monitor

The LAN monitor is currently a subset of UC Davis' Network Security Monitor [3]. The LAN monitor builds its own "LAN audit trail". The LAN monitor observes each and every packet on its segment of the LAN and, from these packets, it is able to construct higher-level objects such as connections (logical circuits), and service requests using the TCP/IP or UDP/IP protocols. In particular, it audits host-to-host connections, services used, and volume of traffic per connection.

Similar to the host monitor, the LAN monitor uses several simple analysis techniques to identify significant events. The events include the use of certain services (e.g., *rlogin* and *telnet*) as well as activity by certain classes of hosts (e.g., a PC without a host monitor). The LAN monitor also uses and maintains profiles of expected network behavior. The profiles consist of expected data paths (e.g., which systems are expected to establish communication paths to which other systems, and by which service) and service profiles (e.g., what a typical *telnet*, *mail*, or *finger* is expected to look like).

The LAN monitor also uses heuristics in an attempt to identify the likelihood that a particular connection represents intrusive behavior. These heuristics consider the capabilities of each of the network services, the level of authentication required for each of the services, the security level for each machine on the network, and signatures of past attacks. The abnormality of a connection is based on the probability of that particular connection occurring and the behavior of the connection itself. Upon request, the LAN monitor is also able to provide a more detailed examination of any connection, including capturing every character crossing the network (i.e., a wire-tap). This capability can be used to support a directed investigation of a particular subject or object. Like the host monitor, the LAN monitor forwards relevant security information to the director through its LAN agent.

An event reported by a LAN monitor is called a network audit record (nar). The record syntax is: nar(Monitor-ID, Source_Host, Dest_Host, Time, Service, Domain, Status).

The LAN monitor has several responsibilities with respect to the creation and use of the NID. The LAN monitor is responsible for detecting any connections related to *rlogin* and *telnet* sessions. Once these connections are detected, the LAN monitor can be used to verify the owner of a connection. The LAN monitor can also be used to help track tagged objects moving across the network. The SSO can also ask for a wire-tap on a certain network connection to monitor a particular user's behavior.

7. The Expert System

DIDS utilizes a rule-based (or production) expert system. The expert system is currently written in Prolog, and much of the form of the rule base comes from Prolog and the logic notation that Prolog implies. The expert system uses rules derived from the hierarchical Intrusion Detection Model (IDM). The IDM describes the data abstractions used in inferring an attack on a network of computers. That is, it describes the transformation from the distributed raw audit data to high level hypotheses about intrusions and about the overall security of the monitored environment. In abstracting and correlating data from the distributed sources, the model builds a virtual machine which consists of all the connected hosts as well as the network itself. This unified view of the distributed system simplifies the recognition of intrusive behavior which spans individual hosts. The model is also applicable to the trivial network of a single computer.

The model is the basis of the rule base. It serves both as a description of the function of the rule base, and as a touchstone for the actual development of the rules. The IDM consists of 6 layers, each layer representing the result of a transformation performed on the data (see Table 1).

The objects at the first level of the model are the audit records provided by the host operating system, by the LAN monitor, or by a third party auditing package. The objects at this level are both syntactically and semantically dependent on the source. At this level, all of the activity on the host or LAN is represented.

At the second level, the *event* (which has already been discussed in the context of the host and LAN monitor) is both syntactically and semantically independent of the source standard format for events.

The third layer of the IDM creates a *subject*. This introduces a single identification for a user across many hosts on the network. It is the subject who is identified by the NID (see section 7.1). Upper layers of the model treat the network-user as a single entity, essentially ignoring the local identification on each host. Similarly, above this level, the collection of hosts on the LAN are generally treated as a single distributed system with little attention being paid to the individual hosts.

The fourth layer of the model introduces the event in *context*. There are two kinds of context: temporal and spatial. As an example of temporal context, behavior which is unremarkable during standard working hours may be highly suspicious during off hours [5]. The IDM, therefore, allows for the application of information about wall-clock time to the events it is considering. Wall-clock time refers to information about the time of day, weekdays versus weekends and holidays, as well as periods when an increase in activity is expected. In addition to the consideration of external temporal context, the expert system uses time windows to correlate events occurring in temporal proximity. This notion of temporal proximity implements the heuristic that a call to the UNIX *who* command followed closely by a *login* or *logout* is more likely to be related to an intrusion than either of those events occurring alone. Spatial context implies the relative importance of the source of events. That is, events related to a particular user, or events from a particular host, may be more likely to represent an intrusion than similar events from a different source. For instance, a user moving from a low-security machine to a high-security machine may be of greater concern than a user moving in the opposite direction. The model also allows for the correlation of multiple events from the same user or source. In both of these cases, multiple events are more noteworthy when they have a common element than when they do not.

The fifth layer of the model considers the *threats* to the network and the hosts connected to it. Events in context are combined to create threats. The threats are partitioned by the nature of the abuse and the nature of the target. In other words, what is the intruder doing, and what is he doing it to? Abuses are divided into *attacks*, *misuses*, and *suspicious acts*. Attacks represent abuses in which the state of the machine is changed. That is, the file system or process state is different after the attack than it was prior to the attack. Misuses represent out-of-policy behavior in which the state of the machine is not affected. Suspicious acts are events which, while not a violation of policy, are of interest to an IDS. For example, commands which provide information about the state of the system may be suspicious. The targets of abuse are characterized as being either *system* objects or *user* objects and as being either *passive* or *active*. User objects are owned by non-privileged users and/or reside within a non-privileged user's directory hierarchy. System objects are the complement of user objects. Passive objects are files, including executable binaries, while active objects are essentially running processes.

At the highest level, the model produces a numeric value between one and 100 which represents the overall *security state* of the network. The higher the number the less secure the network. This value is a function of all the threats for all the subjects on the system. Here again we treat the collection of hosts as a single distributed system. Although representing the security level of the system as a single value seems to imply some loss of information, it provides a quick reference point for the SSO. In fact, in the current implementation, no information is lost since the expert system maintains all the evidence used in calculating the security state in its internal database, and the SSO has access to that database.

In the context of the network-user identification problem we are concerned primarily with the lowest three levels of the model: the audit data, the event, and the subject. The generation of the first two of these have already been discussed; thus, the creation of the subject is the focus of the following subsection.

The expert system is responsible for applying the rules to the evidence provided by the monitors. In general, the rules do not change during the execution of the expert system. What does change is a numerical value associated with each rule. This *Rule Value* (RV) represents our confidence that the rule is useful in detecting intrusions. These rule values are manipulated using a negative reinforcement training method which allows the expert system to continually lower the number of false attack reports. When a potential attack is reported by the expert system, the SSO determines the validity of the report and gives feedback to the expert system. If the report was deemed faulty, then the expert system lowers the RV's associated with the rules that were used to draw that conclusion. In addition to this directed training, which may lower some rule values, the system also automatically increases the RV's of all the rules on a regular basis. This recovery algorithm allows the system to adapt to changes in the environment as well as recover from faulty training.

Logically the rules have the form:

antecedent => consequence

where the antecedent is either a fact reported by one of the distributed monitors, or a consequence of some previously satisfied rule. The antecedent may also be a conjunction of these. The overall structure of the rule base is a tree rooted at the top. Thus, many facts at the bottom of the tree will lead to a few conclusions at the top of the tree.

The expert system shell consists of approximately a hundred lines of Prolog source code. The shell is responsible for reading new facts reported by the distributed monitors, attempting to apply the rules to the facts and hypotheses in the Prolog database, reporting suspected intrusions, and maintaining the various dynamic values associated with the rules and hypotheses. The syntax for rules is:

rule(n ,r ,(single,[A]),(C))).

where n is the rule number, r is the initial RV, A is the single antecedent, and C is the consequence. Conjunctive rules have the form:

rule(n ,r ,(and,[A_1,A_2,A_3]),(C))).

where A_1,A_2,A_3 are the antecedents and C is the consequence. Disjunctive rules are not allowed; that situation is dealt with by having multiple rules with the same consequence.

7.1. Building the NID

With respect to Unix, the only legitimate ways to create an instance of a user are for the user to login from a terminal, console, or off-LAN source, to change the user-id in an existing instance, or to create additional instances (local or remote) from an existing instance. In each case, there is only one initial login (system wide) from an external device. When this original login is detected, a new unique NID is created. This NID is applied to every subsequent action generated by that user. When a user with a NID creates a new login session, that new session is associated with his original NID. Thus the system maintains a single identification for each physical user.

We consider an instance of a user to be the 4-tuple *<session_start, user-id, host-id, time>*. Thus each login creates a new instance of a user. In associating a NID with an instance of a user, the expert system first tries to use an existing NID. If no NID can be found which applies to the instance, a new one is created. Trying to find an applicable existing NID consists of several steps. If a user changes identity (e.g., using UNIX's *su* command) on a host, the new instance is assigned the same NID as the previous identity. If a user performs a remote login from one host to another host, the new instance gets the same NID as the source instance. When no applicable NID is found, a new unique NID is created by the following rule:

```
rule(111,1000,[
  hhar(_,Host1,AUID,_,_,Time1,_,session_start,_,_,'local',_,_,_), /* login */
  \+ (ih(net_user(NID,AUID,Host,_),_,_,_)), /* no NID yet */
  newNID(X)  /* create new NID */
],
  (net_user(X,AUID,Host1,Time1))). /* new net user */
```

The actual association of a NID with a user instance is through the hypothesis *net_user*. A new hypothesis is created for every event reported by the distributed monitors. This new hypothesis, called a *subject*, is formed by the rule:

```
rule(110,100,(and,[
  har(Mon,Host,AUID,UID,EUID,Time,Dom,Act,Trans,Obj,Parent,PID,Ret,Err).
  net_user(NID,AUID,Host,_)
]),
  subj(NID,Mon,Host,AUID,UID,EUID,Time,Dom,Act,Trans,Obj,Parent,PID,Ret,Err))).
```

The rule creates a subject, getting the NID from the net_user and the remaining fields from the host audit record, if and only if both the user-id and the host-id match. It is through the use of the subject that the expert system correlates a user's actions regardless of the login name or host-id.

There is still some uncertainty involved with the network-user identification problem. If a user leaves the monitored domain and then comes back in with a different user-id, it is not possible to connect the two instances. Similarly, if a user passes through an unmonitored host, there is still uncertainty that any connection leaving the host is attributable to any connection entering the host. Multiple connections originating from the same host at approximately the same time also allow uncertainty if the user names do not provide any helpful information. The expert system can make a final decision with additional information from the host and LAN monitors that can (with high probability) disambiguate the connections.

8. Conclusion

Our Distributed Intrusion Detection System (DIDS) is being developed to address the shortcomings of current single host IDS's by generalizing the target environment to multiple hosts connected via a network (LAN). Most current IDS's do not consider the impact of the LAN structure when attempting to monitor user behavior for attacks against the system. Intrusion detection systems designed for a network environment will become increasingly important as the number and size of LAN's increase. Our prototype has demonstrated the viability of our distributed architecture in solving the network-user identification problem. We have tested the system on a sub-network of Sun SPARCstations and it has correctly identified network users in a variety of scenarios. Work continues on the design, development, and refinement of rules, particularly those which can take advantage of knowledge about particular kinds of attacks. The initial prototype expert system has been written in Prolog, but it is currently being ported to CLIPS due to the latter's superior performance characteristics and easy integration with the C programming language. We are designing a signature analysis component for the host monitor to detect events and sequences of events that are known to be indicative of an attack, based on a specific context. In addition to the current host monitor, which is designed to detect attacks on general purpose multi-user computers, we intend to develop monitors for application specific hosts such as file servers and gateways. In support of the ongoing development of DIDS we are planning to extend our model to a hierarchical Wide Area Network environment.

Acknowledgments

The DIDS project is sponsored by the United States Air Force Cryptologic Support Center through a contract with the Lawrence Livermore National Labs.

References

1. Department of Defense, *Trusted Computer System Evaluation Criteria*, National Computer Security Center, DOD 5200.28-STD, Dec. 1985.
2. G.V. Dias, K.N. Levitt, and B. Mukherjee, "Modeling Attacks on Computer Systems: Evaluating Vulnerabilities and Forming a Basis for Attack Detection," Technical Report CSE-90-41, University of California, Davis, Jul. 1990.
3. L.T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc. 1990 Symposium on Research in Security and Privacy*, pp. 296-304, Oakland, CA, May 1990.
4. B. Landreth, *Out of the Inner Circle, A Hacker's Guide to Compuer Security*, Microsoft Press, Bellevue, WA, 1985.
5. T. Lunt, "Automated Audit Trail Analysis and Intrusion Detection: A Survey," *Proc. 11th National Computer Security Conference*, pp. 65-73, Baltimore, MD, Oct. 1988.
6. T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P.G. Neumann, and C. Jalali, "IDES: A Progress Report," *Proc. Sixth Annual Computer Security Applications Conference*, Tucson, AZ, Dec. 1990.
7. T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, H.S. Javitz, A. Valdes, and P.G. Neumann, "A Real-Time Intrusion-Detection Expert System (IDES)," Interim Progress Report, Project 6784, SRI International, May 1990.
8. M.M. Sebring, E. Shellhouse, M.E. Hanna, and R.A. Whitehurst, "Expert Systems in Intrusion Detection: A Case Study," *Proc. 11th National Computer Security Conference*, pp. 74-81, Oct. 1988.
9. W.O. Sibert, "Auditing in a Distributed System: SunOS MLS Audit Trails," *Proc. 11th National Computer Security Conference*, Baltimore, MD, Oct. 1988.
10. S.E. Smaha, "Haystack: An Intrusion Detection System," *Proc. IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, Dec. 1988.
11. H.S. Vaccaro and G.E. Liepins, "Detection of Anomalous Computer Session Activity," *Proc. 1989 Symposium on Research in Security and Privacy*, pp. 280-289, Oakland, CA, May 1989.
12. J.R. Winkler, "A Unix Prototype for Intrusion and Anomaly Detection in Secure Networks," *Proc. 13th National Computer Security Conference*, pp. 115-124, Washington, D.C., Oct. 1990.

Level	Name	Explanation
6	Security State	overall network security level
5	Threat	definition of categories of abuse
4	Context	event placed in context
3	Subject	definition and disambiguation of network user
2	Event	OS independent representation of user action (finite number of these)
1	Data	audit or OS provided data

Table 1. Intrusion Detection Model

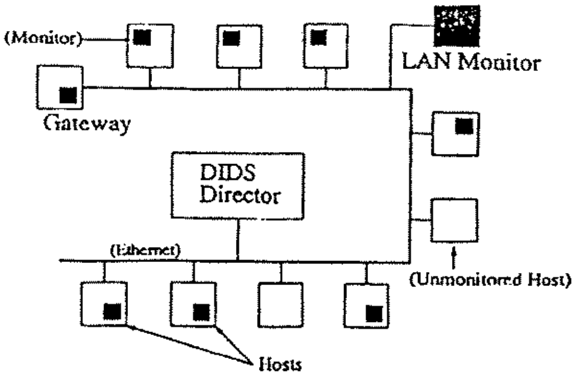


Fig. 1. DIDS Target Environment

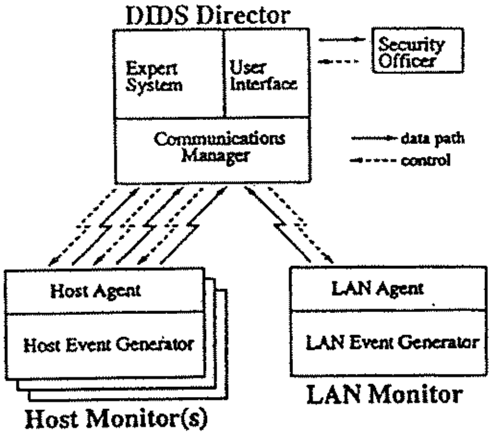


Fig. 2. Communications Architecture

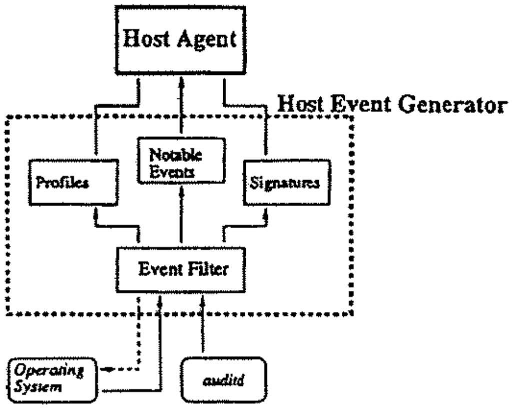
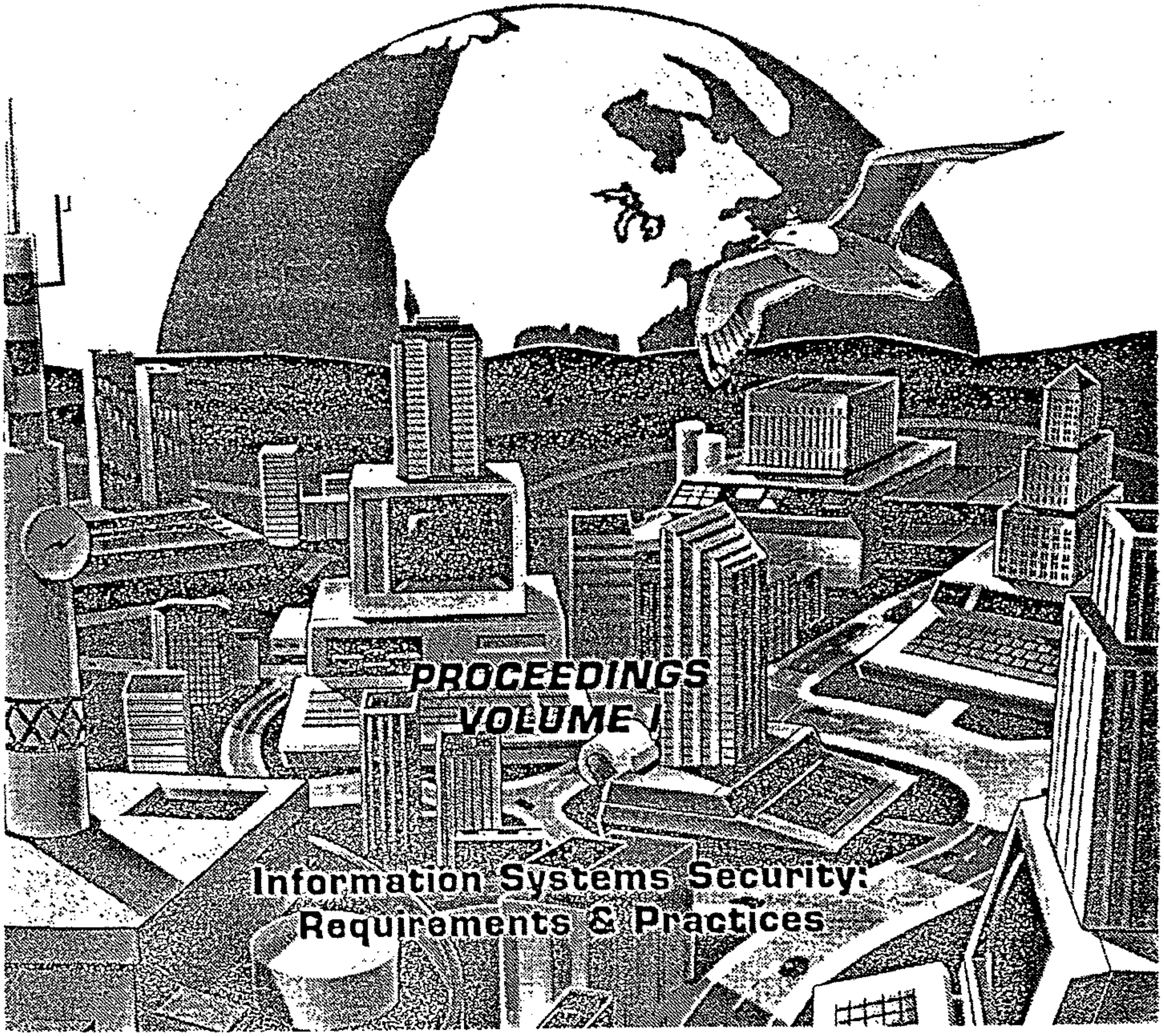


Fig. 3. Host Monitor Structure

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY/
NATIONAL COMPUTER SECURITY CENTER

14TH NATIONAL COMPUTER SECURITY CONFERENCE

October 1-4, 1991
Omni Shoreham Hotel
Washington, D.C.



PROCEEDINGS
VOLUME I

Information Systems Security:
Requirements & Practices